MEMS Extraction

by

Bikram Baidya

A thesis submitted in partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical and Computer Engineering

Date: May 3, 1999

Advisors: Dr. Tamal Mukherjee and Dr. Satyandra K. Gupta
Second Reader: Prof. Gary K. Fedder

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **03 MAY 1999**	2. REPORT TYPE	3. DATES COVERED **00-00-1999 to 00-00-1999**

4. TITLE AND SUBTITLE **MEMS Extraction**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Department of Electrical and Computer Engineering,Pittsburgh,PA,15213-3890**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**42**	

Acknowledgments

I would like to thank my advisors Dr. Tamal Mukherjee and Dr. Satyandra K. Gupta for their constant guidance and encouragement. I am also thankful to Prof. Gary Fedder for his useful suggestions in a number of aspects of the work.

I would like to thank Mr. Sitaraman Iyer and Ms. Qi Jing who helped by providing necessary models for the lumped parameter simulator. I am thankful to Mr. Heeseok Jung for letting me use some of his programs. I'm also thankful to Mr. Hasnain Lakdawala for helping me out with process details and all the other students in the MEMS research group for stimulating discussions from time to time.

I also acknowledge the invaluable support of my family and my friends in Pittsburgh and elsewhere.

Bikram Baidya

This research effort is sponsored by the Defence Advanced Research Projects Agency (DARPA) and U. S. Air Force Research Laboratory, under agreement number F30602-97-2-0323. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the U. S. Air Force Laboratory, or the U.S. Government.

Abstract

Surface micromachined structures are composed of atomic elements like anchors, beams, and fingers, which can further be grouped into functional elements like springs, comb drives and plates. Automatic recognition of these atomic and functional elements is crucial for a structured design methodology for MicroElectroMechanical Systems (MEMS). In the structured design methodology, the schematic design is followed by a transcription of the design into a layout description. Ensuring that the layout description is a correct spatial realization of the schematic requires the extraction of the atomic micromechanical elements. Furthermore, efficient layout verification requires MEMS functional element extraction. An extraction module has been developed which begins with a layout description file and generates the netlist of the schematic corresponding to the layout. An ordinary differential equation solver combined with models of atomic and functional elements can then be used for efficient behavioral verification of the layout by simulating the extracted netlist.

Atomic elements are recognized on the basis of their shape, size and position and are classified into anchors, plate masses, beams, cantilever beams (fingers), joints and holes. This is followed by the extraction of functional elements such as springs, and electromechanical comb sensors and actuators. Comb drives are extracted using similarity in shape, inter-finger gap, electrical connectivity and locality of region of fingers. Springs are detected using a finite state machine type algorithm. A library of springs is written in a library file which is used to generate the graphs needed to match groups of beams and joints in order to recognize a spring. The utility of the extractor is demonstrated for a variety of MEMS devices composed of different types of springs and electrostatic actuators and sensors. Simulation time for the extracted netlist decreased by a factor of 10 when functional element extraction and functional element models were used compared to a netlist of only atomic elements.

Table of Contents

Chapter 1. Introduction

MicroElectroMechanical Systems (MEMS) integrating multi-domain sensors and actuators using conventional microelectronic batch fabrication processes are becoming increasingly complex. In order to design systems with large numbers of multi-domain components, we need to use a hierarchical structured design approach, with design at the schematic level instead of the traditional layout representation used in MEMS design. However, since fabrication can only be done from a layout representation, an automatic or manual layout generation from schematic is necessary. It is essential to be able to translate from the layout representation back to the schematic to reason about layout correctness in meeting the schematic's function as well as to extract geometric parameters for behavioral simulation. An extraction module has been developed which reads in the geometric description of the layout structure and reconstructs the corresponding schematic. This schematic can then be fed to an ordinary differential equation solver or can be compared with the design schematic to validate the correctness of the designed layout. Furthermore, MEMS functional element extraction reduces the size of the simulation problem, enabling efficient design evaluation.

Extraction is commonly used both in the VLSI world and the mechanical world. In VLSI, the main aim of extraction is to detect transistors and calculate resistances, capacitances and inductances in the circuit. The mechanical world on the other hand stresses on detection of geometrical features like holes, slots and other contours. MEMS extraction attempts to detect geometrical features like beams, fingers, holes and also tries to detect electromechanical functional elements like comb drives. VLSI extraction generally relies on information about overlap between layers and gaps between layout areas. Mechanical extraction stresses more on edge detection, pattern matching and using geometrical heuristics. MEMS being born out of both the electrical and mechanical domain uses approaches of both worlds in its extraction. Hence, in MEMS extraction, both layer information and geometrical feature detection is important.

Chapter 2 gives a brief outline of the fabrication processes used in MEMS as well as the design flow and hierarchical design methodology for which the extractor referred to here is best suited. Chapter 3 describes the algorithms used to detect atomic elements in MEMS followed by chapter 4 which describes algorithms used to detect functional elements like springs and comb drives. Chapter 5 presents some results which show the usefulness of the extraction tool followed by chapter 6 which summarizes the present work and briefly describes future directions of research.

Chapter 2. Background

2.1. MEMS Process

There are three major technologies [1][2][3][4] used in MEMS fabrication: bulk micromachining, LIGA and surface micromachining. Within the last decade, surface micromachining techniques have had a phenomenal growth. We will focus on MCNC's Multi-User MEMS Process (MUMPS) [5] due to its simplicity, popularity, and maturity as a surface micromachining process.

An example MEMS device layout is shown in Figure 1. As can be seen in the figure, the device consists of a floating structural layer that is attached to the substrate by anchors. Figure 2 details the process steps that leads to the fabrication of such devices. We focus on how the cross-section A-A' of Figure 1 would look at different points of the fabrication process. First a layer of silicon nitride is deposited on the substrate to form an electrical insulation. This is followed by a layer of polysilicon which is patterned and etched to form electrical interconnects. A sacrificial layer of oxide is then deposited and patterned to get the dimples and the first anchor holes. This stage is shown in Figure 2(a). This is followed by another layer of polysilicon which forms the first structural layer. The photoresist pattern needed to pattern the polysilicon layer is shown in Figure 2(b). Finally the metal layer is deposited and patterned to form the interconnects and pads (Figure 2(c)). The sacrificial oxide layer is etched out and the resulting structure contains the free mechanical device (Figure 2(d)).

The later part of this thesis makes numerous references to various mask layers. We will use the mask conventions laid down in MCNC's MUMPS Design Handbook [5]. Table 1 lists all the layers that we will refer to.

Figure 1: A folded-flexure comb-drive microresonator fabricated in the MUMPS process

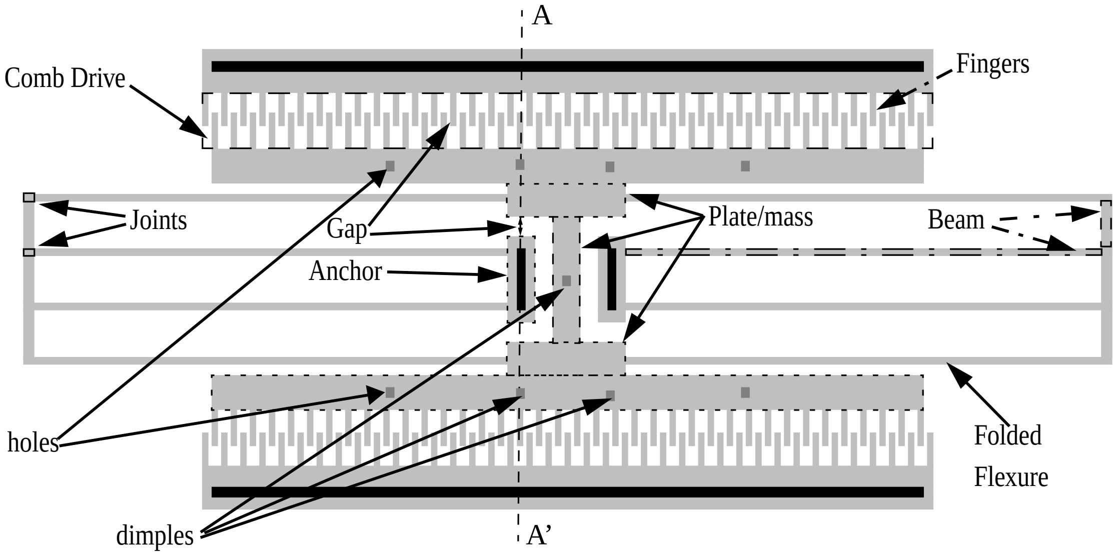


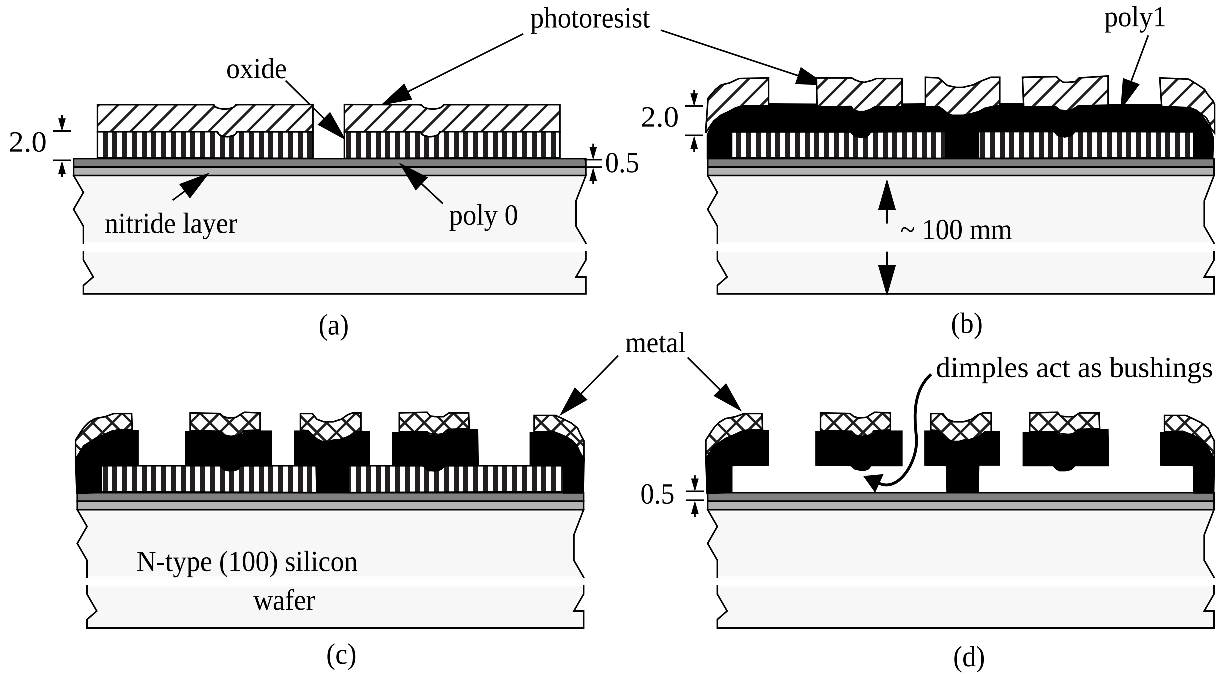
Table 1: Mask conventions used in the MUMPS process

Pnemonic Level Name	Purpose
POLY1	pattern for first structural polycrystaline silicon layer (poly1)
ANCHOR1	open holes for poly1 to nitride or base polycrystaline silicon layer (poly0) connection; elements fabricated after this stage are connected to the nitride/poly0 layer and thus are not floating
HOLE1	provide release holes for poly1 structure. The etchant flows through these holes enabling uniform etching
DIMPLE	create dimples/bushings for poly1 structure

2.2. Design Flow and Design Hierarchy

The current MEMS design methodology (Figure 3) is very cumbersome. It starts off with the designer making a rough sketch of the schematic of the design, shown in the top left of the figure, and very basic equations to ensure feasibility of the design. After being satisfied with the schematic, the designer proceeds to physical layout. At this step, the only tool available to the designer to check the layout is numerical simulation (top right in the figure) using finite element analysis or boundary element analysis. In order to perform such simulation, the layout needs to be meshed properly, which tends to require a lot of time, patience and expertise of the designer [6]. Furthermore, numerical simulation is prohibitively slow and interpreta-

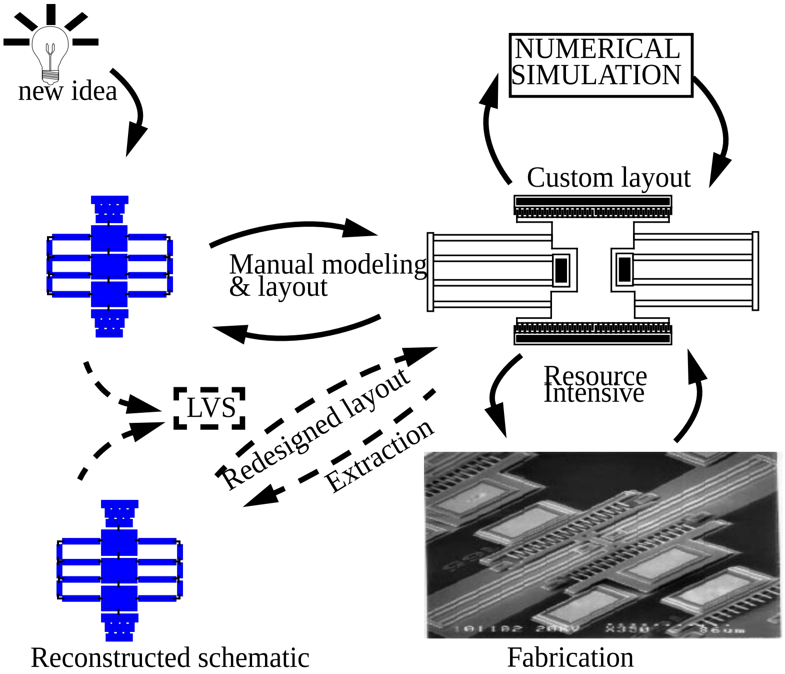
Figure 2: MUMPS process steps highlighting (a) first sacrificial oxide layer, (b) first structural polysilicon layer, (c) metal layer on the polysilicon surface and (d) the final release



tion of the results is tedious. Due to these difficulties, in many cases, the layout is sent for fabrication without proper checking, which sometimes results in non-functional devices. The use of fabrication for design verification (shown in the loop in the bottom right of the figure) is very expensive.

A need for a structured MEMS design process [7][8][9][10], akin to that in VLSI, was felt. This meant that CAD tools were needed at each level of design. The last decade has seen the development of a number of simulation tools based on lumped parameter models [11][12][13][14]. While the top to bottom flow, *i.e.* from design schematic to layout, was being equipped with such tools, nothing much was being done for the reverse flow. Nevertheless, this reverse flow is necessary to verify the designed layout. Our work addresses this problem by developing an extractor and thereby greatly simplifying the task of design checking. By reconstructing the design schematic from layout, the designer will be able to perform faster simulations on the reconstructed schematic and also compare it with the design schematic, thus replacing the fabrication-iteration or the numerical simulation loop with a faster and less expensive schematic-layout loop (shown with broken lines in Figure 3). Hence the extractor lets us exploit the advantages of lumped parameter sim-

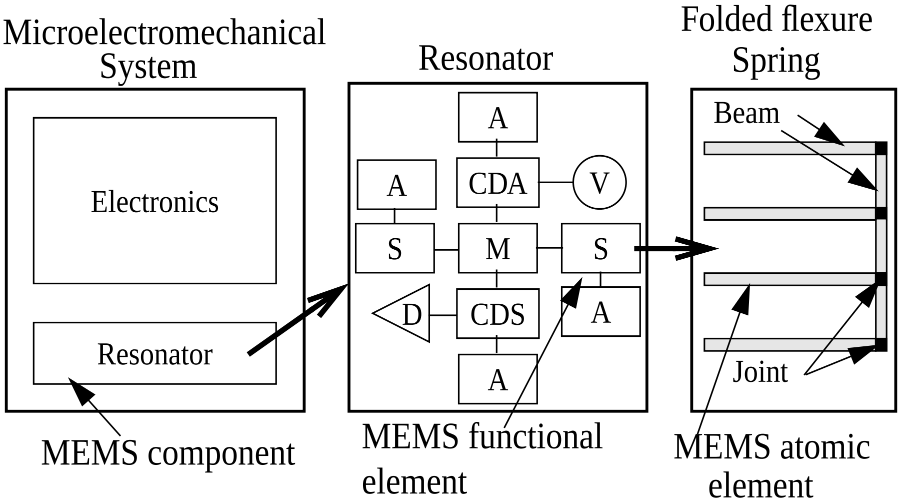
Figure 3: Present and proposed (in dotted line) design flow



ulators in verifying the behavior of the designed layout. Detection of minor errors in the layout, like missing connections, can also be done during the comparison between the reconstructed and designed schematic.

Structured MEMS design methodology requires a hierarchical definition of MEMS components (Figure 4) [10]. A complex suspended microelectromechanical system is composed of electronics as well as MEMS components like resonators, accelerometers and gyroscopes. Each of these MEMS components is in turn composed of functional elements like mass, springs and comb drives. The functional elements can be broken down into much more fundamental or atomic elements like beams, joints, anchors, plate masses and gaps. Following such a hierarchical design methodology allows us to modularize a complex design by using the functional elements as building blocks. This makes it possible to extract and simulate the entire design by extracting each subpart separately instead of the whole design at one time. Hence extraction can be done at each level of hierarchy and the models [14][15] for that hierarchical level can be used for efficient simulation of complex MEMS components.

Figure 4: Hierarchical MEMS design



A: anchor; CDA: comb drive actuator; S: spring; M: mass; CDS: comb drive sensor; V: source; D: detection electronics

Chapter 3. Extraction of Atomic Elements

3.1. Introduction

The extraction process can be broken down to two stages; first being extraction of atomic elements [16] and second being detection of commonly used functional elements [17]. Figure 1 highlights both the atomic and functional-level features that can be extracted from a MEMS layout. This chapter looks at the extraction of atomic elements in much more detail. Since layout design is a reflection of the designer's style, even if two designers do the same design, the final layouts might differ. This poses a serious bottleneck for recognition. To overcome this problem, we convert the given layout to a representation which is unique for a given design. Feature-based recognition is then used to detect the various atomic elements. The final recognized set is then optimized to reduce the total number of nodes required to represent it as a netlist. Using the information contained in the recognized set we may either generate a netlist, which can then be compared with the original design netlist, or continue for functional element extraction.

While the key idea has been borrowed from the extraction process used in VLSI world [18][19][20][21], there are a number of differences. Unlike VLSI, in MEMS, the shape, size and position of an object is of utmost importance and plays a crucial role in deciding what kind of element it is. The data structure that is most opted for in VLSI extractor designs [22][23] is a list of all non-vertical edges, sorted first according to their abscissa followed by their ordinates and lastly by their slopes. These edge-based data structures make overlap detection simple but are not computationally useful for shape detection. Instead we use a list of polygons which eases the task of shape recognition. Since our detection loops are to be run for all the polygons, it was found unnecessary to use other complex data structures like quad trees or binning which are generally used at places where detailed positional information is needed at all parts of the layout. The only positional information needed in MEMS extraction is about neighboring polygons and this is easily solved by maintaining pointers to the neighbors. In effect our representation is a hybrid of linked lists and corner-stitching [24].

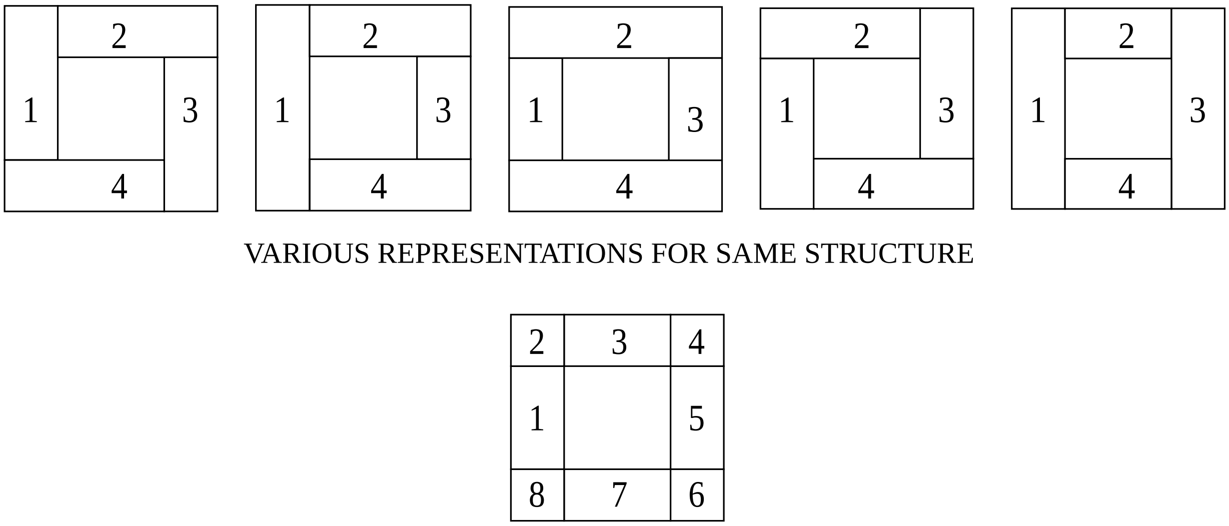
In this chapter, we first describe the representation that we use to achieve uniqueness. This is followed by definitions of the atomic elements we wish to extract and the steps through which extraction is actually done. Finally, we give a detailed description of some of the algorithms used for atomic element extraction.

3.2. Canonical representation

We define the canonical representation of the layout to be the one which uses minimum number of rectangles to cover the given layout area, such that infinitesimal outward extensions of an edge of any rectangle never intersects with the interior of the layout area. We use the term *layout area* to define the area which represents the actual component in the layout, *i.e.*, it is the interior area(s) defined by the boundary/boundaries of the geometrical representation of the component in the layout. Thus, in the canonical representation, the layout is made up of small rectangles such that each rectangle has *at most one neighbor per edge and each edge is either fully covered by a neighbor or not covered at all*. This can be easily achieved by extending the boundary edges into the interior of the layout area till it meets another boundary edge. The resulting representation uniquely partitions the layout area.

As a vast majority of MEMS layouts are Manhattan, this thesis refers to Manhattan designs only. Hence our task is to canonize polygons whose edges lie along one of two orthogonal coordinates. Figure 5

Figure 5: Canonical representation



VARIOUS REPRESENTATIONS FOR SAME STRUCTURE

CANONICAL REPRESENTATION

explains our idea of canonical representation. We further assume that the input is in the form of rectangles, *i.e*, a rectangle cover for the layout is supplied to us. Given this as input, we proceed to canonize the structure using the algorithm described below. The hierarchical description of the chip (written, perhaps, in CIF, *i.e.* Caltech Intermediate Form) is first flattened and then the strucutural pattern in the first polysilicon layer is canonized.

The primary interaction in the canonization process takes place between two sets; the input set and the output set. The output set will eventually contain the canonical version of the input set. The output set is always kept in canonical state with respect to its contents. Elements from the input set are selected sequentially and added to the output set. Whenever there is an addition to the output set, its equilibrium might be destroyed (*i.e*, the output set might no longer be a canonical set). If this occurs, a series of operations is initiated which ultimately brings the output set back to its equilibrium or canonical state. This is repeated till the input set is emptied, and, at this point, the output set will contain the canonical representation of the input layout. The process which drives the output set to equilibrium, after it is disturbed by the insertion of a new element, is described in Figure 6.

Figure 6: Algorithm to canonize a layout

```
1. Let R be the input set.
2. Let G be the output set initialized to a NULL set.
3. Let us randomly pick an element r from R and initialize a working set P by adding r to P.
4. Q = {x| ADJ(x, r); x is an element of G}
   ADJ(x, r) is an operator which returns those elements x (x is an element of G), which are adjacent to r
5. for all q in set Q
     for all p in set P
       $V_q$ = set of vertices of Q
       $E_p$ = set of edges of P
       if CON($V_q$,$E_p$) then SPLIT(p,q)
       Function CON($V_q$, $E_p$) returns TRUE if there exists a pair v in $V_q$ and e in $E_p$ such that v lies on or is
       contained by e. Function SPLIT(p, q) splits p by the edges of q.
6. for all p in P
     for all q in Q
       if CON($V_p$, $E_q$) then SPLIT(q,p)
7. while(Q!= NULL)
     Q’ = {x|ADJ(x, q), x is an element of G}
     for all q in Q
       for all q’ in Q’
         if CON($V_q$, $E_{q’}$) then
           SPLIT(q’, q) and
           Q =Q’
8. G = G $\cup$P
9. R = R - {r}
10. if R!= NULL then go to step 3
    else end
```

In a global sense, the algorithm partitions as well as determines the neighbors of each rectangle in the final canonized state. This neighbor information is obtained by comparing each rectangle being added to the output state with the rectangles already in the output state. This neighbor information is saved for later use in the recognition algorithms. Since this neighbor search is done for each of the rectangles in the final canonized representation, the algorithm has an asymptotic upper bound of $O(n^2)$ where n is the number of rectangles in the final canonized representation.

The procedure used to obtain a canonical representation for Manhattan layouts can easily be extended to non-Manhattan designs which use polygons. The key idea of developing the canonical representation sequentially, by extending boundary edges of the representative blocks, can be used for polygons also. The final set will then consist of polygons which have *at most one neighbor per edge such that no edge is par-*

tially covered by an adjacent polygon. This technique can also be extended to layouts containing arcs where the tangent to the curve or the arc itself can be extended inside the layout area.

3.3. Atomic elements

The current extraction module has been implemented with the goal of extracting inertial MEMS components. Such components are made up of anchors, masses, beams, joints and fingers. This section defines each of these atomic elements functionally and geometrically.

An anchor is a region of the MEMS structure that is constrained in its movement. It provides support to the suspended regions of the MEMS component. It may also provide electrical connectivity between parts of the same component. It is difficult to detect anchor areas of a layout from geometric description of the structural layer alone. Information from non-structural layers are therefore used to extract anchor location in a given layout. For the MUMPS process the areas of anchor in a layout are easily found from the ANCHOR1 layer information. This layer defines the anchor cut in the layout and the extractor uses its information to recognize anchor rectangles in the canonized representation of the structural layer.

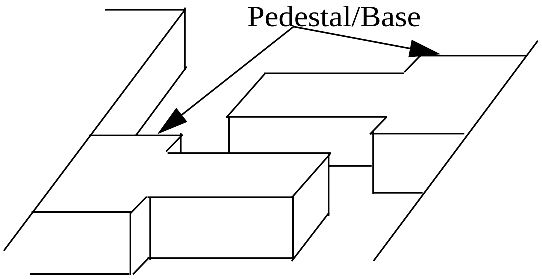
The mass region of a MEMS structure is defined to be the rigid suspended region of the layout. The space between the mass and the surface to its side and below determines the amount of gaseous damping. Ideally mass elements are large enough to be considered to be rigid. In order to have such large floating areas, it becomes necessary to have holes in the layout so that the etchant can successfully release the mass area. In addition, bushings (Figure 1, Figure 2(d)) are often used to prevent large mass plates from sticking to the substrate (a negative side of the wet etching release step). Such process-specific information is used by the extractor to detect mass areas. In the MUMPS process the HOLE1 layer provides information about the location of holes and DIMPLE layer provides information regarding the location of the bushings. The extractor also detects the absence of the structural layer as holes, and uses them as hints for mass areas.

Beams are non-rigid suspended regions of the MEMS component and govern the strucutural compliance in different directions of motion. Since they are designed to be flexible, beams are generally thin and long and connected only at their shorter sides. Behaviorally, beams posses all properties of mass element with the exception that they are not rigid. The spring constant of an inertial MEMS component is governed by the shape and placement of beams in the structure. Geometrically, beams are rectangles that have neighbors only on their two shorter edges. Thus the extractor uses neighbor information to recognize beams in a given layout.

Joints connect two or more beams structurally. They can be considered to be small mass areas which help to change the orientation and placement of two or more physically connected beams. Behaviorally joints are similar to beam elements, but since they are generally very small, their contribution towards the spring constant of the component on the whole can be modeled in the adjacent beams, leaving the joint as a logical connectivity element. Geometrically they are rectangles which only have beams as their neighbors.

Fingers are floating cantilever beams in the layout. They are generally used to increase the capacitive area in electrostatic actuators and sensors. Two overlapping and electrically isolated sets of such fingers are normally used to design a comb drive [25], one of the most popular electrostatic functional element used in MEMS designs. A more detailed description of comb drives is given later. In addition to their contribution towards electrical behavior of the MEMS component, fingers also contribute significantly towards mass and damping of an inertial structure. Geometrically, fingers are rectangles having only one neighbors on one of the shorter edges. Designers sometimes design fingers with pedestals (Figure 7) [26]. Such pedestal-based of fingers result in an inter-finger gap that is less than that allowed by the process rules. For a comb drive made of such fingers, the normal gap between the rotor and stator fingers is constrained by the minimum gap allowed by the process rules. When the fingers engage, the effective gap between a finger and its opposite pedestal is reduced. Hence the sensitivity of the comb drive is increased without increasing the number of fingers. The thin cantilever part of a pedestal finger is connected to the pedestal on one of the shorter sides of the pedestal with the other short side of the pedestal connected to either a mass or an anchor.

Figure 7: Fingers with pedestals



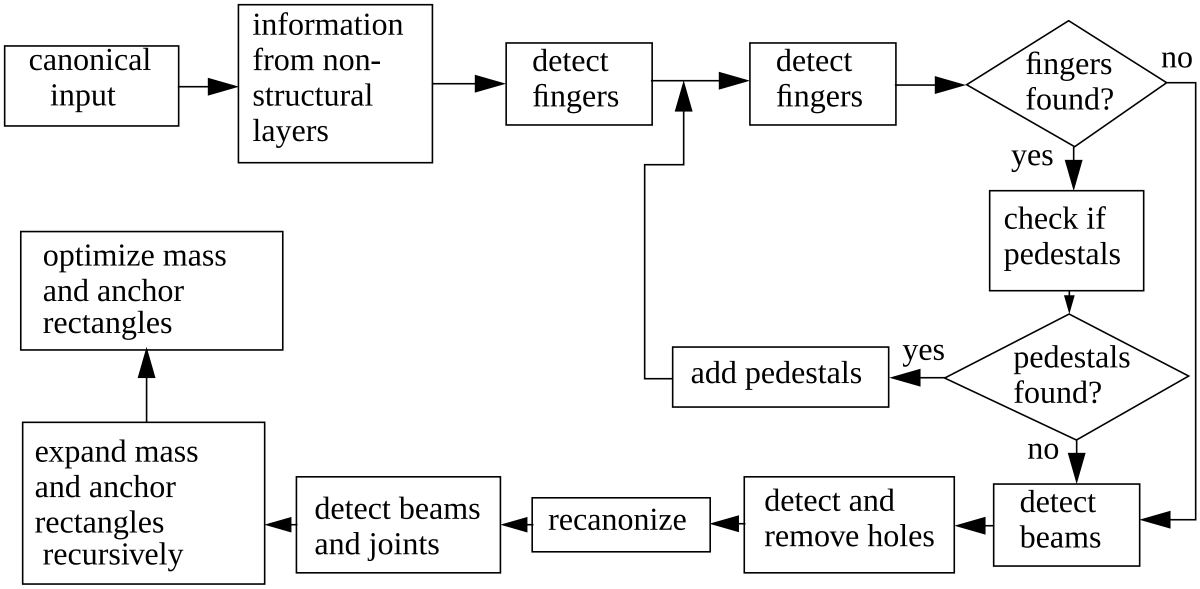
3.4. Extraction flow for atomic elements

In our analysis of the algorithms used for the recognition, we will use *n* to symbolize the number of unrecognized rectangles in the canonical representation of the layout at that particular step. The main steps of the recognition process are shown in Figure 8. The order in which the different atomic elements are recognized is important because it affects both the geometric heuristics used for recognition, as well as the speed of the recognition algorithms.

The first step in the detection is to use the information from the non-structural layers to detect mass and anchor areas. This calls for boolean operations, like *OR* and *AND*, of different layers. The *AND* of ANCHOR1 layer and structural layer provides information about the location of anchor areas in the layout. The *AND* of the structural layer with the *OR* of HOLE1 and DIMPLE layer provides us with potential locations of mass elements. The information from this step is separately maintained and is used to mark out mass and anchor areas whenever the layout is recanonized.

The next step is to recognize the fingers. During the process of canonization, an edge extended from another rectangle may split a finger lengthwise. Such split fingers will not be detected by the normal finger detection routine, *i.e.* only by checking if a rectangle has only one neighbor on one of its shorter side. A proximity test is used to detect such split fingers. This test works on the assumption that in a particular set of fingers at least one finger is not split and will be detected in the first pass, which is done in *O(n)* time. The proximity test for split fingers is done in the second pass. If a collection of adjacent rectangles are such that each of the constituent rectangles have at least one short edge with no neighbor and lies in the proxim-

Figure 8: Steps in extraction of atomic elements



ity of a number of other fingers of similar dimension and if these fingers have the same bound (along abscissa if the fingers lie along the ordinate or vice versa) as that of the concerned set of rectangles, then the rectangles are merged together and marked as a finger. Hence, the algorithm performs check between the fingers already detected (*m*) and the remaining rectangles in the canonized set and thus takes *O(mn)* time.

In the process of canonizing the layout, it can be shown that the presence of fingers (as in a comb drive) attached to plate or anchor tend to divide the layout unnecessarily. To prevent these unnecessary partitions in the canonization process from adversely affecting the speed of feature-based recognition algorithms, we separate out the fingers after they are detected. The removed fingers are stored as a separate group. Next the remaining rectangles are merged to get a simplified cover of the remaining layout. This merging procedure can be done in *O(n)* time because the neighbor information is already available. The subroutine to make a canonical representation is run on this modified cover to get a new canonical layout representation. Though the canonization algorithm takes quadratic time, the number of rectangles being considered here is considerably less and the resulting recanonized set results in a great improvement in the speed of operation.

In order to detect pedestal-type fingers (Figure) [26], a pedestal detection loop is used. After a set of fingers have been detected and removed, a second finger search on the recanonized layout detects the pedestals (since they are topologically equivalent to cantilever beams once the fingers are removed). An adjacency check for each of these prospective pedestals with the already removed first set of fingers is used to confirm them as pedestals. The pedestal recognition loop has a time complexity of *O(n + mp)* where *m* is the number of fingers detected in the previous step and *p* is the number of fingers detected in the current loop. Since fingers must either attach to a mass or anchor, the base of a finger (or the pedestal, if it is a pedestal type finger) is marked as potential mass or anchor.

After having recognized the fingers, we proceed to recognize physical holes in the structural layer. An initial beam recognition is also done so that gaps between beams are not recognized as holes. The holes in the layout are the floating rectangles, that are obtained from a maximal horizontal representation of the *NOT* of the structural layer, which do not have any beams as their neighbor in the original layout. Holes detected at this step are replaced by mass rectangles in the original layout. We keep track of total increase in area due to such holes and also due to the holes from HOLE1 layer and, when we calculate the area of the plate, we delete this excess virtual area. This information is also annotated in the netlist generated as mass factor for the mass. In addition, other physical parameters like centre of mass and moments of inertia are also calculated and written in the netlist file.

The next step is to detect the beams and joints which are very easily detected from the neighbor information alone. This is followed by recognition of rest of the mass and anchor rectangles. In order to recognize the rest of the mass and anchor areas, we use the rectangles that have been already marked as mass/anchor and recursively expand them in all directions such that all unrecognized rectangles that can be reached from these pre-recognized rectangles are appropriately marked. The expansion process checks each rectangle only once and hence runs in linear time. At this stage we have the recognized version of the layout which can be now used for detection of functional elements.

3.5. Analysis of utility algorithms used in extraction of atomic elements

The *OR* routine operates on the canonical representation of two layers and results in a new *OR* layer. The first step is to perform inter layer canonization between the two input layers. This takes $O(n_1*n_2)$ time where n_1 and n_2 are the number of rectangles in the two layers, respectively. This procedure is followed by a merger of the two sets of rectangles. The duplicate pairs of the layout can now be found using box overlap check. Each such check takes constant time and since it is run on each and every rectangle, the time required for this step is $O((n_1+n_2)^2)$. A maximal horizontal representation of the remaining layout is then done in order to reduce the final number of rectangles in the resulting *OR* layer. In order to do such a merging, neighbor information is found for the entire *OR* layer. This takes quadratic time and can be combined with the step where duplicate pairs are found. The speed of the *OR* routine can be improved by maintaining region limits which would reduce the time required for overlap check.

The *AND* routine uses the same approach as the *OR* routine. After a merged set of fully canonized representations of the two layers is obtained, we check for duplicate rectangles and store them in the *AND* layer. The *AND* layer is also merged to get a maximally horizontal representation and hence it also takes $O((n_1+n_2)^2)$ time.

The *NOT* routine takes in a single input layer and generates a new *NOT* layer. First the vertical and horizontal boundary edges of the input layout are sorted separately. A binary tree is used for sorting the edges which in best case will result in an upper bound of $O(nlgn)$, but in worst case will take $O(n^2)$ time. The operation can be made $\theta(nlgn)$ always by using B-trees but was avoided in the current implementation because the rest of the *NOT* algorithm takes $O(n^2)$ time. The bounding box of the given input layout is used as the layout area which is then partitioned using the edges in descending order of their coordinate value. The reason for such a strategy is because such a partitioning process allows us to always split the left bottom corner rectangle of the layout area. This helps eliminate the time required to find the region which a particular edge will split. First, the area is partitioned using the horizontal edges in linear time and then the vertical edges are used to split the bottom rectangle. Each split in the bottom rectangle is propagated in the top *n* rectangles and hence the whole splitting operation takes quadratic time. Next a box overlap check is done between the rectangles in the split layout area and the rectangles in the uncanonized representation of the input layout. The purpose of using the uncanonized representation is to reduce the time required since the time required at this step is the product of the number of rectangles in the two sets. The rectangles in the split area that overlap with those in the input layout are removed. Finally we have a split version of the *NOT* layout on which a linear merge operation can be done to get a maximally horizontal representation.

3.6. Final optimization of the mass and anchor rectangles

The canonical representation results in a great number of mass and anchor rectangles. Since the lumped parameter models for mass normally treats the mass elements as simple rigid bodies, there is not much gain in defining different models for different mass shapes. Nevertheless, having large number of mass elements increases the number of nodes and slows down the lumped parameter or behavioral simulator. Similarly a large number of anchor rectangles unnecessarily increases the size of the netlist.

In order to remove the problem, we combine the mass rectangles (and likewise the anchor rectangles) that are actually a part of a single plate, to minimize the number of nodes necessary for the simulation of the extracted MEMS device. We use an approach similar to that used in corner stitching [24]. In corner stitching the rectangles are first expanded horizontally, *i.e.*, adjacent rectangles having the same vertical coordinates are combined, followed by vertical expansion. Thus the representation is maximally horizontal. In our case we still stick to these discrete combining steps (*viz.* vertical and horizontal expansion) but try to keep the sequence which results in a lesser number of rectangles. This is done by comparing the horizontal-then-vertical expansion with the vertical-then-horizontal expansion. The sequence of steps used to achieve this is described in Figure 9 and runs in linear time because the neighbor information needed in step 4 of the algorithm is already present as an outcome of the canonical algorithm.

Figure 9: Optimization algorithm

```
1. initialize A = NULL and B = NULL
    where, A and B are sets which will contain the final merged structure.
2. take an element g from the set G and initialize set P with it. Where G is the set containing the elements that make
   up the canonical structure and P is the working set.
3. delete the element g from set G
4. for all p in P
     Q = {x| x = NBR(G,p)
     where NBR(G, p) returns the neighbors of p in G
5. for all q in Q
     if TYPE(p, q) then
          $P = P \cup \{q\}$ and
           $G = G - \{q\}$
     Function TYPE(p,q) returns a value TRUE iff p and q are of the same type.
6. a = HOR(P)
    where HOR(x) returns a set which is the horizontal merged version of set x.
7. a = VERT(a)
    where the VERT(x) function returns the vertical merged version of set x.
8. b = VERT(P);
9. b = HOR(b);
10. $A = A \cup a$
11. $B = B \cup b$
12. if G!= NULL go to 2
13. if N(A) > N(B) then
     output A
      else
     output B
      where N(C) stands for the cardinal number of a set C
```

Chapter 4. Extraction of Functional Elements

4.1. Introduction

As MEMS designs gather maturity, designers tend to design MEMS systems consisting of an increasing number of MEMS components. The netlist for such systems become large because of the large number of nodes. Simulation time for most schematic level simulators are highly dependent on the number of nodes. In order to reduce the simulation time, it is preferred to describe the schematic using lumped parameters for functional elements instead of a netlist composed of only atomic elements. Hence, it is advantageous to extract functional elements instead of just stopping at the atomic element level. The most important functional elements in inertial systems are electromechanical comb transducers and springs. This chapter describes the various types of such functional elements and the algorithms that are used for extracting them.

4.2. Electromechanical comb actuators

Silicon microstructures have long been actuated and sensed electrostatically by means of fixed electrodes forming parallel-plate capacitors with the structure. The main drawback of these structures have been that the electrostatic force is nonlinear unless the movement is small compared to the electrostatic gap. The need for a linear drive/sense device led to the design of the electrostatic comb [25] which consists of interdigited cantilever beams called *fingers*. One side of the comb is fixed (stator) while the other side is allowed to move (rotor). Such a device can be used to drive the device as well as sense motion in the device. Any harmonic motion of the rotor can be sensed by the current i_s which is given by

$$i_s = V_S(\delta C/\delta x)(\delta x/\delta t) \qquad (1)$$

where V_S is the bias voltage and x gives the position of the rotor. For the comb structure, $\delta C/\delta x$ is constant depending only the distance between the comb fingers. At the drive port, the displacement x can be given as a function of the drive voltage by

$$x = F_x/k_{sys} = V_D^2(\delta C/\delta x)/(2*k_{sys}) \qquad (2)$$

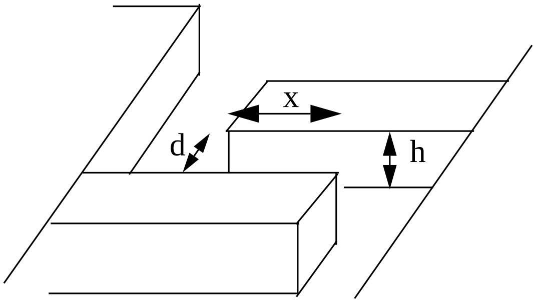
where F_x is the electrostatic force, k_{sys} is the spring constant of the system and V_D is the drive voltage. If $V_D = V_P + v_d\, sin(\omega t)$ then

$$\delta x/\delta t = (\delta C/\delta x)/(2*k_{sys}) *[2\omega V_P v_d cos(\omega t) + \omega v_d^2\, sin(2\omega t)] \qquad (3)$$

which again is linear since $\delta C/\delta x$ is constant.

Mechanical design of such comb drives can be of two types: linear and torsional. In the linear design, the comb fingers lie parallel to the direction of motion and can be used to excite and sense motion parallel

Figure 10: A pair of cantilever beams forming the building block of electrostatic comb actuator/sensor



to the plane of substrate. The torsional design is for torsional resonant plates in which the comb fingers lie on arcs of concentric circles and can excite and sense torsional motion about the center of these circles.

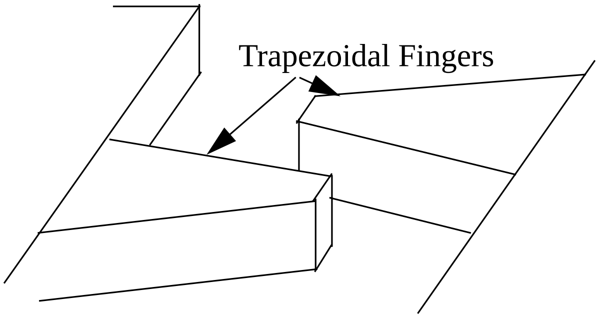
Both the sense current (when the comb is used as a sensor) and the displacement (when the comb is used for excitation) are dependent on $\delta C/\delta x$. Using normal equations for a parallel plate capacitor we get,

$$\delta C/\delta x = \varepsilon nh/d \qquad (4)$$

where n is the number of capacitors formed by the interdigited comb fingers, h is the vertical height of the fingers and d is the distance or gap between two fingers (Figure 10). Thus, to improve the sensitivity of the device, the gap d must be made as small as possible. Limitations of fabrication technology do not allow designers to reduce this gap beyond a certain point and this creates a limit to the sensitivity that can be achieved. After meeting this limit, the sensitivity can be increased by increasing the number of comb fingers (thereby increasing n) at the cost of increased weight and area. To overcome this restriction, a pedestal design for comb fingers [26] is sometimes used (Figure 7).

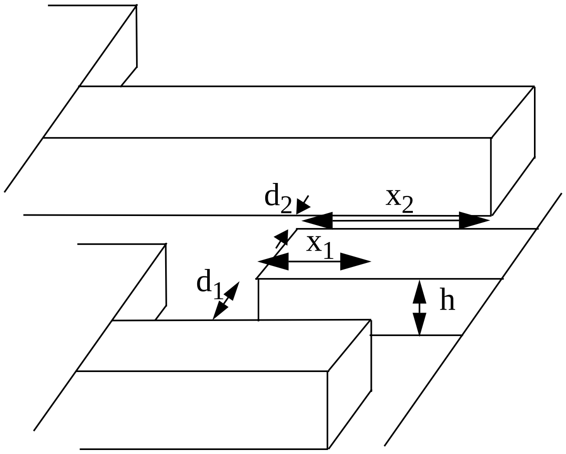
All the comb drives described above have $\delta C/\delta x$ constant resulting in a linear response. Sometimes quadratic response is required and in such cases we need a $\delta C/\delta x$ which is linear with x. A trapezoidal comb finger (Figure 11) is used for such purposes. Here the gap between two finger changes linearly with x and hence results in a quadratic response. Such combs are used in very special devices where a controlled nonlinear excitation is required.

Figure 11: Trapezoidal comb fingers



The comb drive structures described above were found to suffer from levitation problems [27]. This causes a decrease in the actual area of overlap with the increase in drive voltage and δ*C*/δ*x* no longer remains linear. Though this effect is very small and is negligible for normal operations, it does play an important role in devices where linearity is very crucial. One way to solve the levitation problem is to eliminate the ground plane and remove the substrate beneath the structures. Another way is to have a top ground plate suspended above the comb drive. These arrangements achieve a balanced vertical force on the comb. Both of these solutions require complicated fabrication sequences. An easier solution is to reverse the polarity on alternating drive fingers resulting in an altered field distribution where the potential distribution along the z-axis is constant or nearly constant. Various structures have been developed to alternate the polarity at every stationary drive finger (Figure 12), every other finger, every forth finger and so on, depending on the amount of error correction needed. Such a structure is also used to sense transverse motion via differential sensing of the different sets of capacitances.

Figure 12: A set of fingers in a differential comb drive



The extraction module is able to recognize most of the above mentioned comb drive structures. The only comb designs which presently cannot be detected are the torsional comb drives and combs having trapezoidal fingers. This limitation is due to their non-Manhattan nature. The present implementation works only for Manhattan designs. Another assumption made in the extraction module is that the fingers belonging to one comb are aligned on the same base which is either parallel to the horizontal or vertical axis. This assumption is found to be true for most general designs unless the designer wants a comb drive which rests on an inclined plane. Such designs are uncommon though not impossible. The interfinger gap in a single comb is generally uniform but in very special designs the gap on two sides of a finger may not be the same. The present comb detection module detects such variation and prints out a warning message. Thus, the present comb drive detection module can detect functional combs which fall within the scope of Manhattan designs.

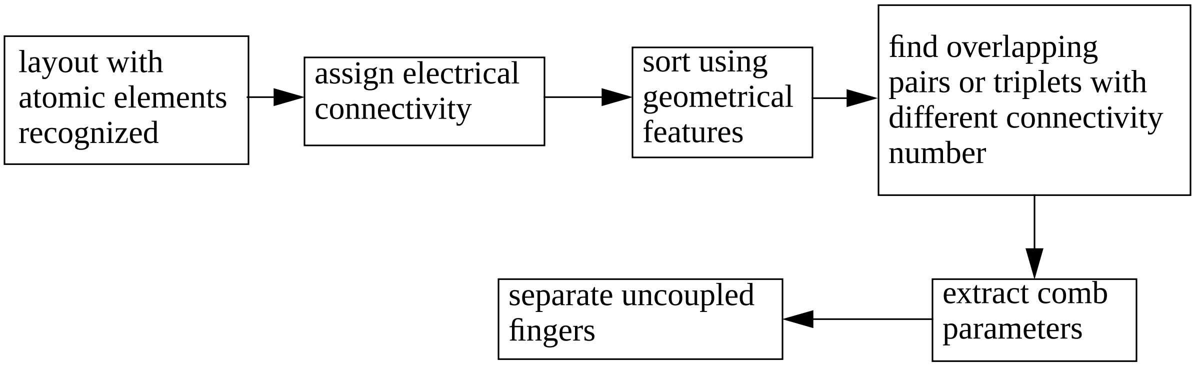
4.3. Comb drive extraction

The comb drive extraction process is shown in Figure 13. It starts with a connectivity analysis of the set of recognized fingers. Fingers having electrical connectivity are given the same connectivity number. Fingers are then sorted into buckets based on their orientation. The sorted fingers are stored in a linear linked list and hence in worst case the sorting takes $O(n^2)$ time, where n is the number of fingers. The speed can be improved by using more advanced data structures like heaps and B-trees but were not used in the current implementation because the time requirement for this step was not found to be crucial for the extraction flow.

Each such finger bucket is then checked for uniformity of the fingers with respect to region of occurrence, length of fingers, width of fingers and inter-finger gap. If the fingers have pedestals, then the region of occurrence, length and width of the pedestal, inter-pedestal gap and the relative position of the pedestal with respect to the thin cantilever finger are also checked. The buckets are partitioned whenever any non-uniformity is found in any of these parameters. A box cover of each of the buckets is then created. The implementation of each of these steps takes linear time since each finger/pedestal is visited only once at each step.

The box covers are checked mutually for overlap using box overlap rules in $O(n^2)$ time, where n is the number of such box covers. Whenever an overlapping pair is found between two buckets having different connectivity numbers, they are matched in size and combined to form a comb drive. The matching function takes care that there are no uncoupled comb fingers in the final comb drive. If one of the overlapping sets have comb fingers which do not couple capacitively with any of the fingers of the other set, then the finger set is partitioned so that the final pair only contains coupled fingers. Overlapping triplets are also detected and checked to see whether they form differential finger comb drives. Another criterion to be satisfied for such triplets is that two of them must be connected to anchors. In such a case, the rotor set is matched with each of the stator sets to get the final matched triplet, which are then merged to form a comb drive. The matching function goes through the two sets being matched finger by finger and hence runs in $O(n)$ time where n is the total number of fingers in the two sets.

Figure 13: Comb drive extraction


layout with atomic elements recognized
assign electrical connectivity
sort using geometrical features
find overlapping pairs or triplets with different connectivity number
extract comb parameters
separate uncoupled fingers

4.4. Mechanical springs

Springs are composed of beams and joints and connect the suspended plate mass to the anchors. The design a of proper spring possesses a lot of challenges because it controls motion of the suspended structure. Proper selection of springs often results in motion in one preferred direction. Spring design also affects cross axis coupling and primarily defines the eigenmodes of motion of the components. Hence, there are numerous springs that can be designed. Through time, some springs have become standard and here we discuss a few of them.

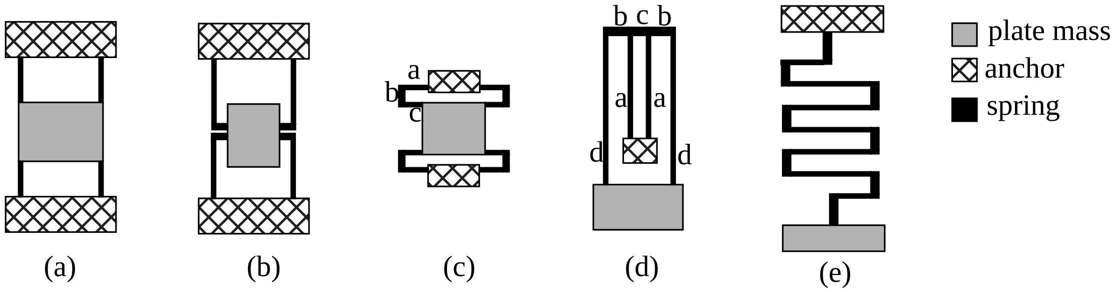
The simplest out of all springs is a fixed-fixed flexure (Figure 14(a)). It consists of a simple straight beam connecting the suspended mass to the anchor and has a very stiff spring constant because of extensional axial stress in the beams. Normally, springs are placed symmetrically in the design in order to reduce abnormalities like cross axis coupling. Hence, in the figure the mass is connected to the anchor using four fixed-fixed flexures. Such a spring will have more flexibility in the horizontal direction and will be stiff in the vertical direction. Its flexibility in the z-direction will depend on the thickness of the beams.

Crab-leg springs and U-springs (Figure 14(b) & (c)) are modifications to the fixed-fixed beam so as to reduce peak stress in the flexure at the cost of reduced stiffness in undesired directions. The crab leg (Figure 14(b)) is composed of two separate beam segments which may differ in length and width. The relative stiffness of the spring in the horizontal and vertical directions will depend on the dimensions of the two beams. The U-spring (Figure 14(c)) is composed of three separate beams which also may differ in length and width from each other. Normally, it is designed to give more flexibility in the horizontal direction but the spring has lot more flexibility in the vertical direction also because of the horizontal beam in the middle. Both the springs have reduced stress because residual stress can be released by minor deformation of the beams of the springs which have at least one side that is not fully fixed.

A meander spring (Figure 14(e)) is also a modified version of a fixed-fixed flexure which helps achieve more compliance using less space. The preferred direction of flexibility depends on which set of beams is longer. The meander spring is more flexible in the direction orthogonal to the set of beams that are longer in length. In addition to the lengths and widths of different types of beams that constitute the meander spring, the number of loops in the spring also determines its spring constant in different directions.

A folded flexure (Figure 14(d)) also reduces axial stress and gives more compliance while occupying less area and is often the preferred spring for inertial designs. Though there are different designs for folded flexures, the one shown in Figure 14(e) is the one that is most commonly used.

Figure 14: Normally used springs: (a) fixed-fixed, (b) crab leg, (c) U-spring, (d) folded-flexure, (e) meander spring



4.5. Spring extraction

Spring detection is done using a Finite State Machine (FSM) based algorithm. Unlike conventional FSMs, the algorithm may have more than one final state whose simultaneous satisfaction is necessary for the outcome of the recognition to be true. The FSM can be defined by *M = {S, L, U, G, F}*, where

S = start state;

L = language = {*joint, beam, NULL*};

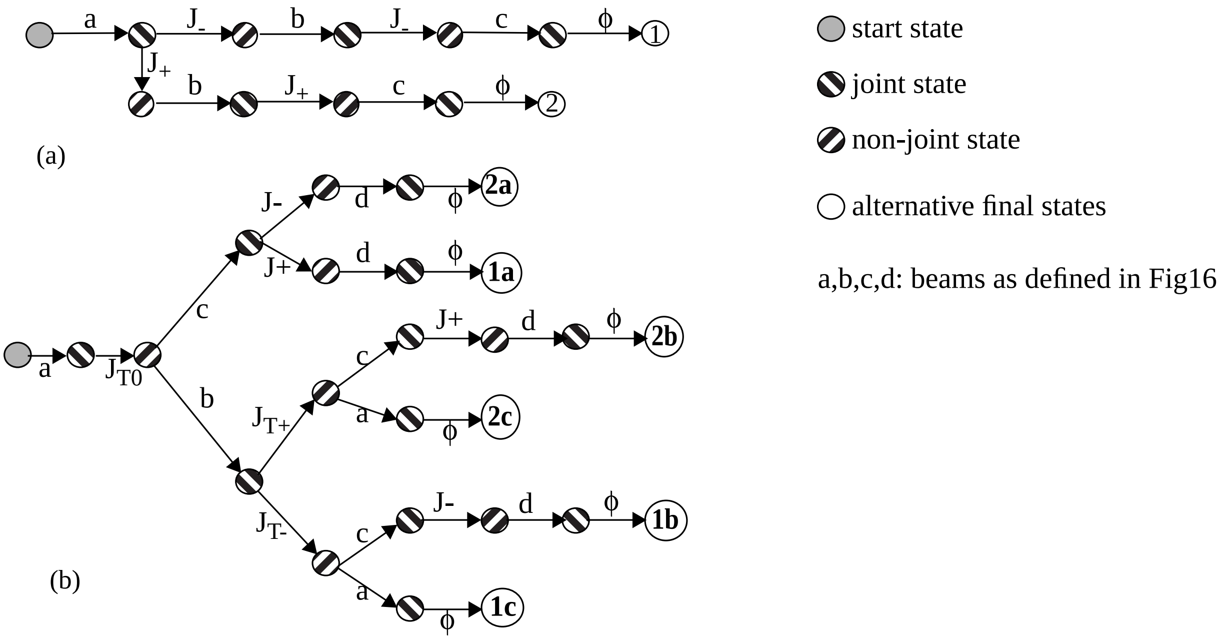
U = transition states which are either joint-state (states which accepts either joints or NULL) or non-joint-state (state which accepts only beams);

G = the set of rules for the FSM; and

F = set of final states.

A joint is defined to be a node having one input port and at most three output ports and is labelled using the '*m*' (from moment) and '*t*' (from transition) parameters. The *t*-parameter is 1 only if there is an output port along the direction of the input port. An output port at right angles to the input port contributes a +1 or -1 to *m*-parameter depending whether the twist direction is anticlockwise or clockwise. The six types of joints possible using such a convention are shown in Table 2. The set of beams for the language depend on the spring to be detected. For example, a U-spring requires three beams (Figure 14(c)) which may or may not be equal in dimension, while a folded flexure requires four type of beams which must be arranged as shown in Figure 14(d). The graphs which will be used by the FSM to recognize U-springs and folded flexures are shown in Figure 15 (a) and (b) respectively. The final states 1 and 2 in Figure 15(a) are the two

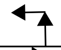
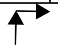
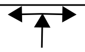
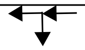
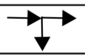
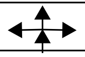
Figure 15: FSM for (a) U-spring and (b) folded flexure



alternative possibilities for a U-spring, one turning in clockwise direction and other in anticlockwise direction. The graph for the folded flexure also has two such alternatives but for each there are three conditions (a, b, c) to be satisfied simultaneously. This is because the folded flexure has four ports, one of which will act as the input port and the others as output ports. Since a folded flexure has two anchor points, there may be two ways in which the same folded flexure may be traversed and hence the two alternatives (1 and 2).

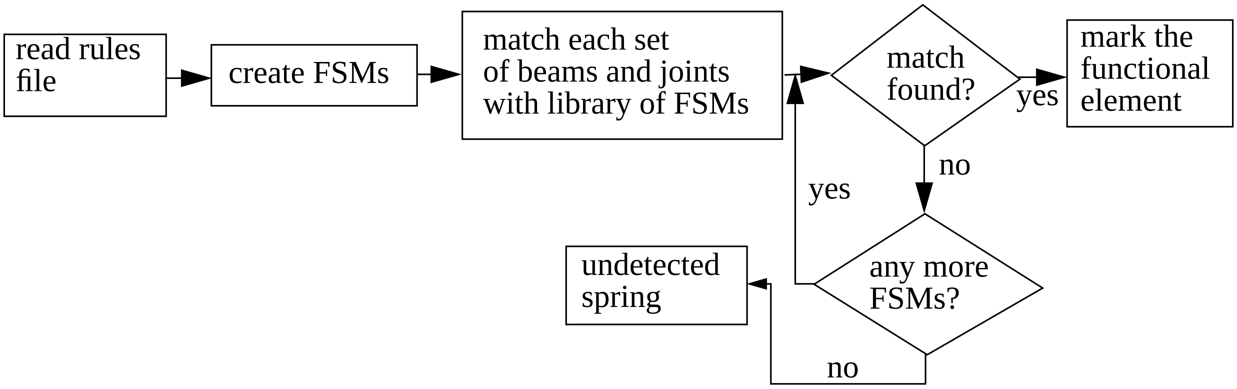
The FSM for each of the springs is created by reading in its description from the library file. The connected sets of beams and joints obtained after the atomic recognition is then passed through each of these FSMs to recognize their type. For each such set, the input is started from a beam which is connected to an

Table 2: Dictionary of joints

Joint name	m - param	t- param	ports	example
J_+	+1	0	2	
J_-	-1	0	2	
J_{T0}	0	0	3	
J_{T+}	+1	+1	3	
J_{T-}	-1	+1	3	
J_0	0	+1	4	

anchor rectangle. The flow of the spring detection algorithm is shown in Figure 16. The overhead of setting up the library of springs is linear with respect to the number of states in the springs defined in the library. The detection part of the algorithm takes *O(mn)* time, where *m* is the total number of states in all the spring libraries and *n* is the total number of beams and joints in the given layout. The spring extraction algorithm can be extended to detect springs made of smaller springs. For example, a crab leg may be made up of two serpentine springs connected at right angles. For such cases, the S-State must be modified to take in springs and beams and the detection loop must be run in loops until there is no more merging of springs. The present implementation does not attempt such a detection because such springs do not seem to be feasible in practice and also deriving models for such springs is quite complex. Nevertheless the implementation has scope of expansion to do such a detection, if needed, in future.

Figure 16: spring extraction


read rules file
create FSMs
match each set of beams and joints with library of FSMs
match found?
yes
mark the functional element
no
yes
any more FSMs?
undetected spring
no

Chapter 5. Results

5.1. Introduction

This section shows a select few results to demonstrate the capability of the current extractor which implements the algorithms discussed in the previous chapters.

5.2. Folded flexure resonator

Figure 17 shows the result for a folded flexure resonator. The input is shown in Figure 17(a). It can be clearly seen that in case of the beam marked in the figure, the rectangle representing it penetrates into the mass area. This shows the need for a canonical representation. Figure 17(b) shows the rectangles in the canonical representation. It is important to note that each constituent rectangle has only one neighbor on each side. This helps us in deriving the neighbor information very easily. As can be easily seen, the number of rectangles in this representation has increased tremendously. This occurs due to extension of all edges and the situation is made severe due to the presence of fingers. Thus, in the next step, we separate out the fingers, after having recognized them, and then recanonize the remaining layout. The result is shown in Figure 17(c). This step brings a huge reduction in the number of rectangles. We then proceed to apply our feature recognition algorithms to recognize beams. Also, inter-layer interaction information is used to recognize some of the mass and anchor rectangles. The result of these feature recognition algorithms is shown in Figure 17(d). This step is followed by the expansion of the mass and anchor rectangles to detect the remaining rectangles resulting in the complete recognition of all the rectangles as shown in Figure 17(e). The next step is to reduce the number of rectangles needed to represent the mass and anchor area. Figure 17(f) shows a minimal representation where the rectangles have been first merged horizontally and then merged vertically. It can be seen that the number of rectangles needed to represent the central I-shaped mass here is three. In case we had opted for vertical merge first followed by horizontal merge, then the number would have been five. Thus the algorithm has taken the correct decision in selecting between maximal horizontal and maximal vertical representations.

The resulting recognized set can then be used to recognize functional elements or can be used to generate a netlist composed of only atomic elements. Figure 17(g) shows the layout after the functional elements have been recognized. The extracted netlist was simulated using lumped parameter models [14][15] for the functional elements and the simulation result is shown in Figure 17(h). The simulation was found to be more than 10 times faster than when it was simulated using a netlist made of models of only atomic elements.

5.3. Accelerometer

Figure 18(a) shows an accelerometer which uses a differential comb drive to detect motion in the horizontal direction. A meander spring was used to act as the suspension device for the mass. Figure 18(b) shows the layout with the atomic elements recognized and Figure 18(c) shows the layout after the functional elements have been recognized. The extracted netlist was simulated for transient behavior and the result for a 1g acceleration pulse is shown in Figure 18(d)&(e). The extraction algorithm detects and removes the holes in the mass of the layout so as to reduce the number of nodes in the final netlist.

5.4. Gyroscope

Figure 19(a) shows a three-fold symmetric gyroscope which uses U-springs and beams for its suspension mechanism and uses pedestal type fingers in its comb drive for increased actuation. Figure 19(b) shows the final extracted layout. The extracted netlist could not be simulated because the model for the pedestal type comb drive is not complete.

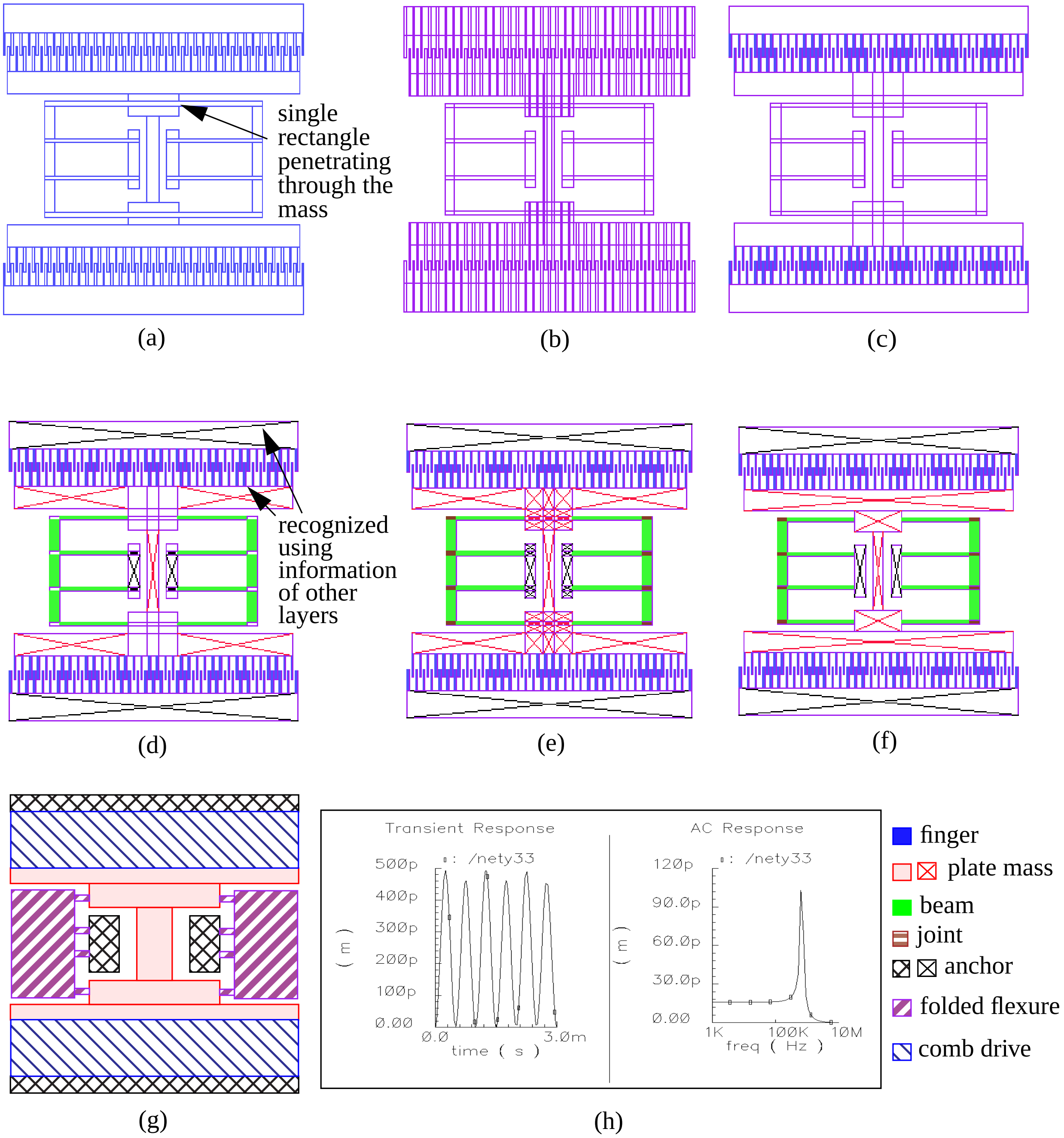
5.5. Four resonators in four directions

Figure 20(a) shows a layout consisting of four resonators in four directions. The extracted netlist for it is shown in Figure 20(b). As can be seen, the optimization algorithm resulted in optimized number of mass rectangles in the extracted netlist. Thus, for the resonators with comb drives in the vertical direction, the algorithm chose a maximal horizontal merging for the mass rectangles and for the other two it selected maximal vertical representation. The whole layout is extracted by the tool in less than few seconds.

5.6. Erroneous resonator layout

The usefulness of the extractor is demonstrated in this example (Figure 21) where the input layout of a folded-flexure resonator was found to have a very small error which was not detected by the human eye. When extracted, the netlist gave two sets of comb actuators instead of just one pair. On inspecting the orig-

Figure 17: Folded flexure resonator; (a) layout, (b) canonical representation, (c) canonical representation after separating the fingers, (d) intermediate state, (e) detected state, (f) optimized result, (g) functional element extraction, (e) transient (1KHz source) and ac (resonant frequency = 691.8KHz) analysis of extracted netlist



(a)

(b)

(c)

(d)

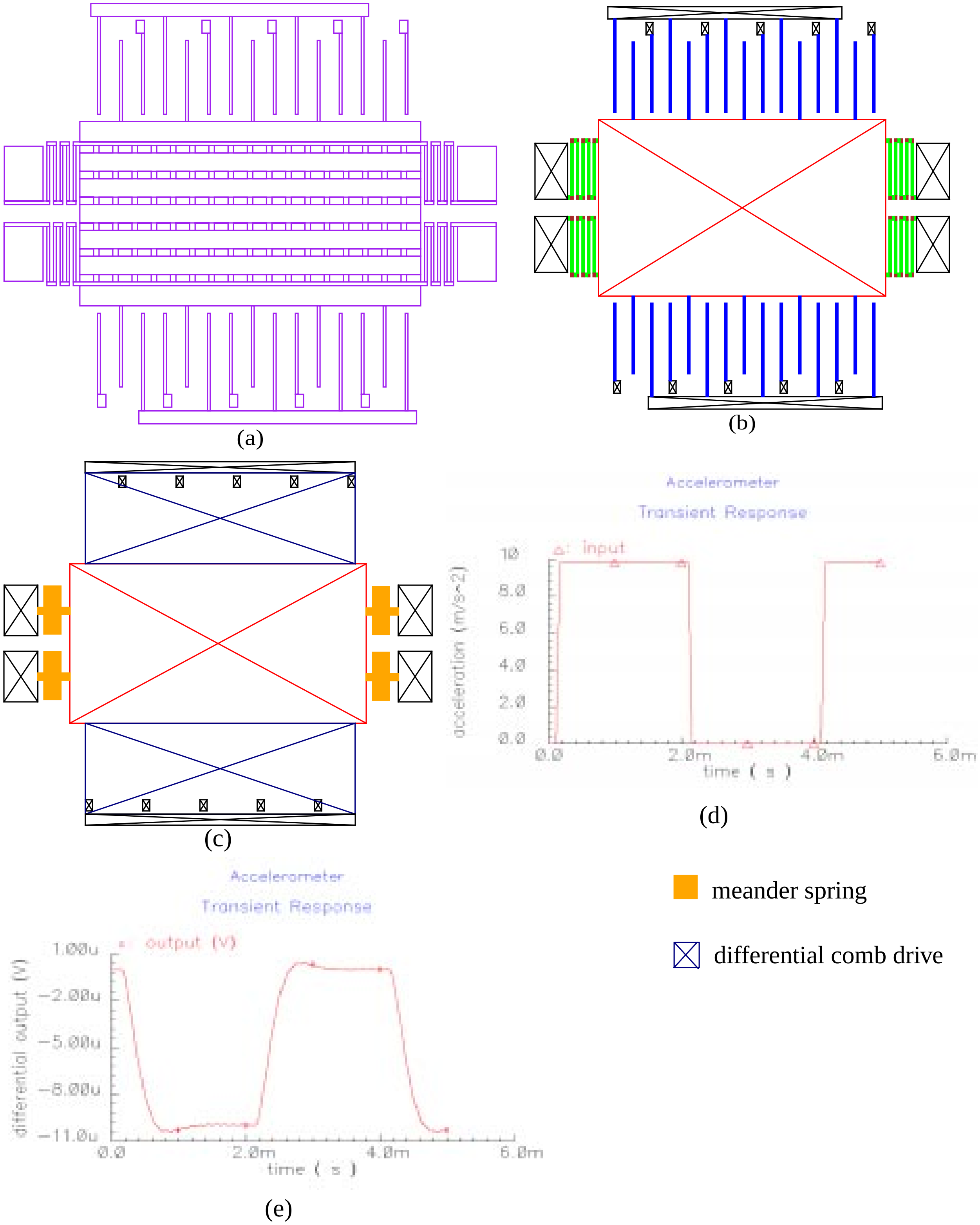
(e)

(f)

(g)

(h)

Figure 18: Accelerometer using a meander springs and differential comb drive; (a) input layout, (b) layout with atomic elements recognized, (c) functional elements extracted, (d) input to the schematic, (e) transient response of the accelerometer for a one g pulse acceleration



(a)

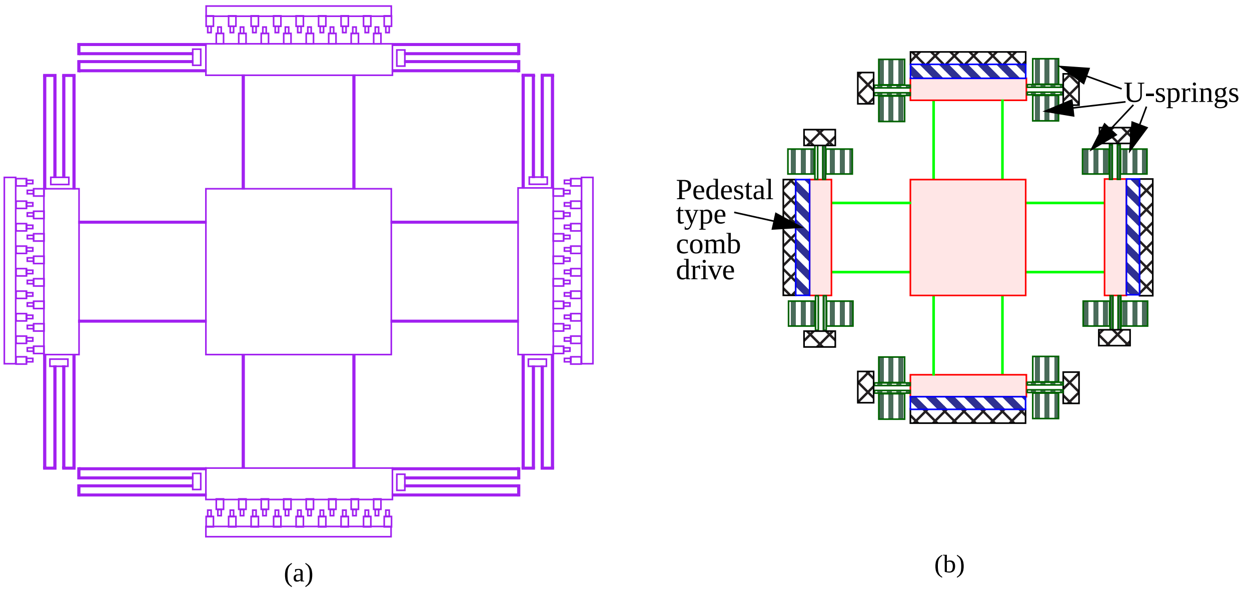
(b)

(c)

(d)

(e)

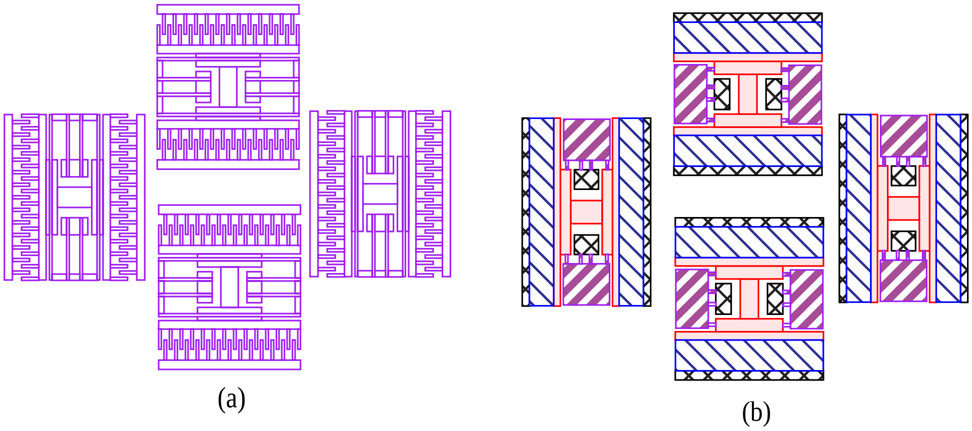
Figure 19: Three-fold symmetric gyroscope; (a) input layout, (b) extracted netlist



(a)

(b)

Figure 20: Orthogonally placed resonator sets; (a) input layout, (b) extracted netlist

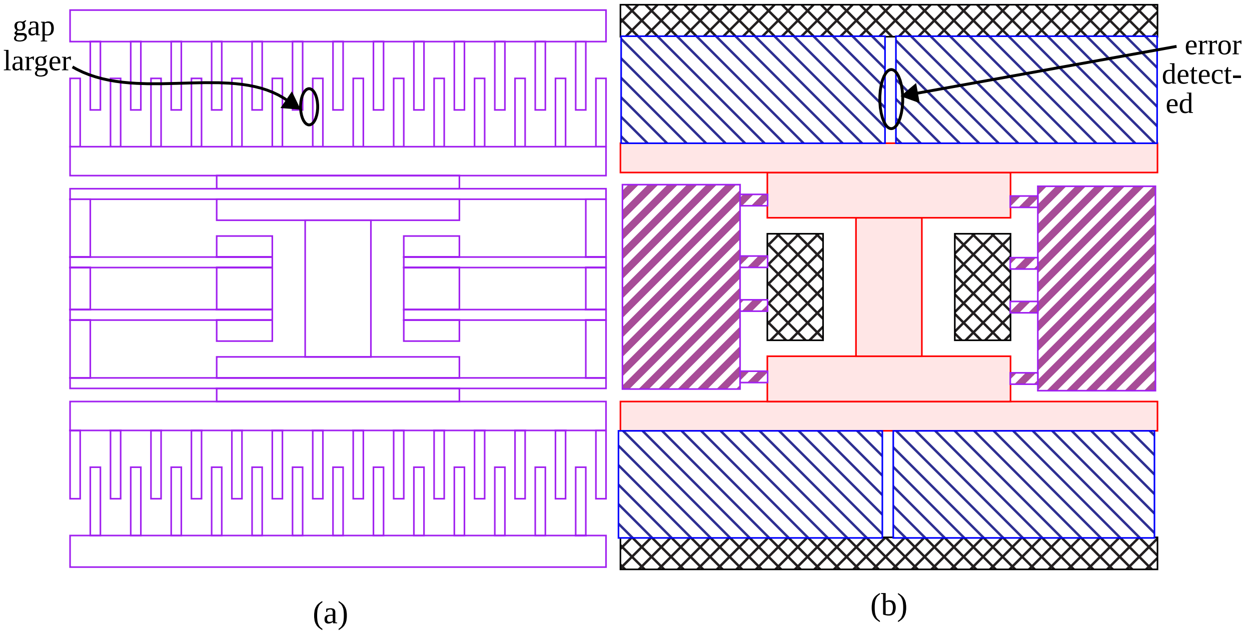


(a)

(b)

inal layout, it was found that there was a difference in the gaps between the two halves of each comb drive. This was because when the half was being replicated and placed to double the size of the comb actuator, a small human error resulted in a gap which was more than the gaps between other fingers. This was detected by the extractor and was interpreted as two sets of comb actuators. the layout was then corrected to remove the error.

Figure 21: Erroneous layout of a resonator; (a) input layout, (b) extracted layout



5.7. Erroneous accelerometer layout

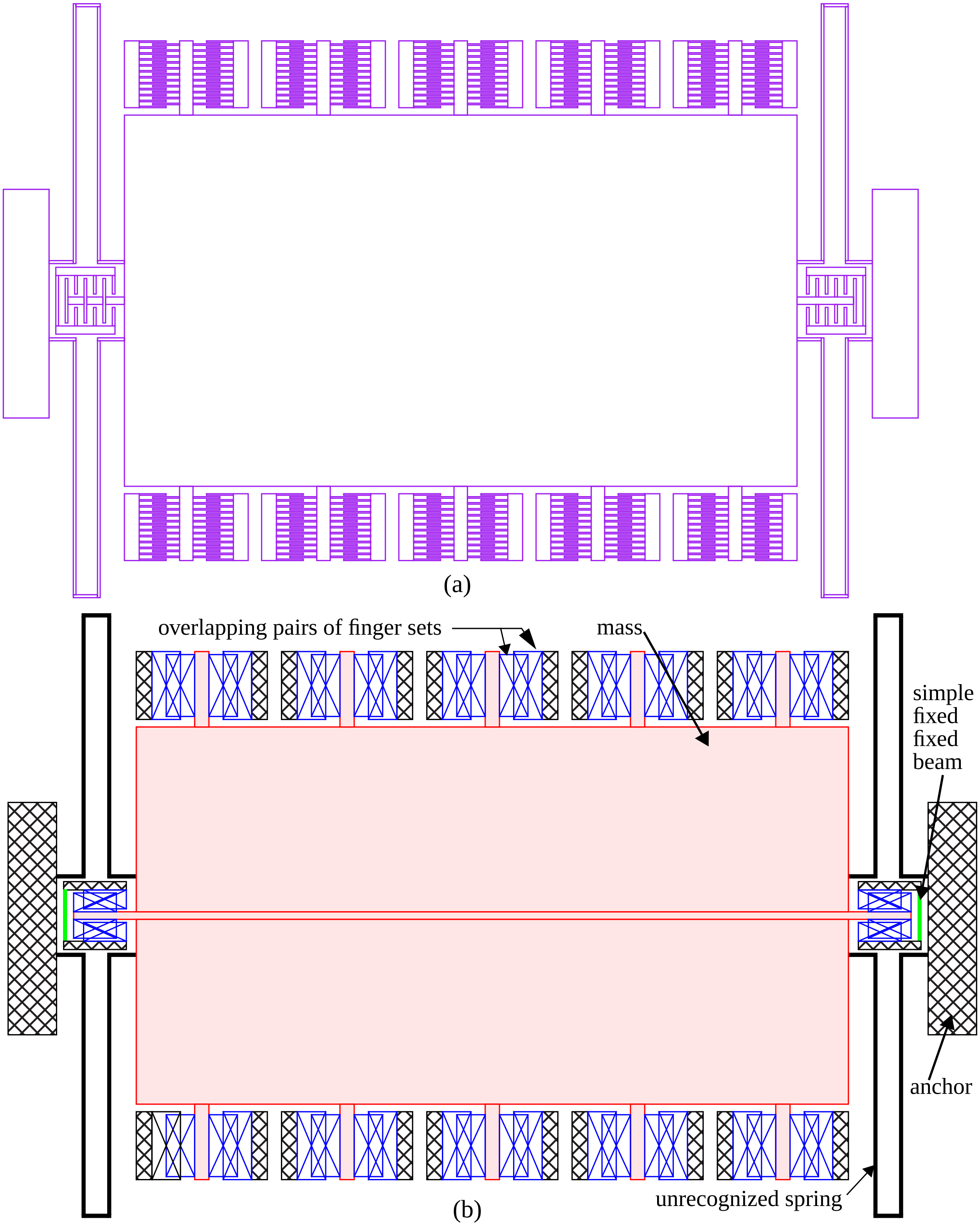
In another example, the extractor was used on an accelerometer layout (Figure 22). The extracted netlist gave overlapping sets of fingers but did not combine them to form comb drives. On inspection, it was found that the POLY0 layer was missing from the layout. Thus all the comb finger sets had the same electrical connectivity number. Hence, the extractor did not merge the pairs of sets of comb fingers to form comb drives. The layout was then corrected and the POLY0 layer added at appropriate places.

Chapter 6. Conclusion and future work

6.1. Conclusion

In order to verify that the detailed design of a MEMS device is a correct spatial realization of the desired schematic representation, we need capabilities to reconstruct schematic representations from the spatial representation of the device. Reconstructed schematics can be used to identify design problems without performing expensive physical prototyping. In this work, we present a feature-based methodology for reconstructing schematics from the layout information. Reconstructed schematics provide information

Figure 22: Erroneous accelerometer; (a) input layout, (b) extracted netlist



(a)

(b)

regarding types and parameters of various constitutive elements, and interconnectivity of various elements. Our current implementation is limited to designs based on the MUMPS process utilizing Manhattan geometry. The extraction is performed to the functional element level which leads to fewer elements in the extracted netlist. We expect that our tool will help the CAD designers to shrink their design time and also help them in building much more complex MEMS designs.

6.2. Future work

The extraction work can be extended in various directions. One direction of improvement is to extend the tool to handle non-Manhattan designs and make it process independent. The final aim is to have an extractor which can take in a process descriptor file and an element (both atomic and functional) descriptor file and use this information to extract any layout in that process. The definitions of canonical representation and most of the heuristics used for extraction can be easily modified for polygonal designs but layouts having arcs possess a big challenge for a generalized extractor. The present recognition is directed towards extraction of inertial devices. Extending the extraction capability to other types of devices like fluidics also holds potential for future work.

REFERENCE

[1] Geppert, L, “Solid state [semiconductors. 1999 technology analysis and forecast],” *IEEE Spectrum*, Jan. 1999, pp. 52-6.

[2] J. Bryzek, K. Petersen and W. McCulley, “Micromachines on the march,” *IEEE Spectrum*, May 1994, pp. 20-31.

[3] R.T. Howe, *et. al.*, “Silicon Micromechanics,” *IEEE Spectrum,* July 1990, pp. 29-35.

[4] W.C. Tang, “Overview of Microelectromechanical Systems and Design Processes,” *34th DAC Proceedings,* 1997, pp. 670-3.

[5] D.A. Koester, R. Mahadevan, K.W. Markus, *Multi-User MEMS Processes (MUMPs) Introduction and Design Rules*, available from MCNC MEMS Technology Applications Center, 3021 Cornwallis Road, Research Triangle Park, NC 27709, rev. 3, Oct. 1994, 39 pages.

[6] H. Lakdawala, B. Baidya, T. Mukherjee and G.K. Fedder, “Intelligent Automatic Meshing of Multilayer CMOS Micromachined Structures for Finite Element Analysis,” *Proc. of MSM ‘99,* San Juan, Puerto Rico, pp. 297-300, April 19-21, 1999.

[7] E.K. Antonsson, “Structured Design Methods for MEMS,” *NSF sponsored workshop on Structured Design Methods for MEMS,* November 12-15, 1995.

[8] G.K. Fedder, “Structured Design Methods for MEMS: Essential Tools for RApid MEMS Development,” *NSF sponsored workshop on Structured Design Methods for MEMS,* November 12-15, 1995.

[9] T. Mukherjee and G. K. Fedder, “Structured Design Of Microelectromechanical Systems,” *Proceedings of the 34th Design Automation Conference (DAC ‘97)*, Anaheim, CA, June 9-13, 1997.

[10] G. K. Fedder, “Structured Design of Integrated MEMS,” *Proc. of MEMS ‘99,* Orlando, Florida, Jan. 17-21, 1999, pp.1 -8.

[11] I. Getreu, “Behavioral Modelling of Analog Blocks using the SABER Simulator,” *Proc. Microwave Circuits and Systems,* pp 977-980, August 1989.

[12] J.E. Vandameer, M.S. Kranz and G.K. Fedder, “Nodal Simulation of Suspended MEMS with Multiple Degrees of Freedom,” *1997 International Mechanical Engineering Congress and Exposition: The Winter Annual Meeting of ASME in the 8th Symposium on Microelectromechanical Systems,* Dallas, TX, Nov. 16-21, 1997.

[13] J.E. Vandemeer, M.S. Kranz, G.K. Fedder, “Hierarchical Representation and Simulation of Micromachined Inertial Sensors,” *Proc. of MSM*, Santa Clara, LA, Apr. 6-8, 1998 pp. 540-545.

[14] G.K. Fedder, Q. Jing, “NODAS 1.3 - Nodal Design of Actuators and Sensors,” *IEEE Transactions on Circuits and Systems (II) Special Section on BMAS* (to be published).

[15] S. Iyer, Y. Zhou, T. Mukherjee, “Analytical Modeling of Cross-axis Coupling in Micromechanical Springs,” *Proc. of MSM ‘99*, San Juan, Puerto Rico, pp. 632-35, April 19-21, 1999.

[16] B. Baidya, S.K. Gupta, T. Mukherjee, “Feature-Recognition for MEMS Extraction,” *CDROM Proc. 1998 ASME DETC*, Atlanta, GA, Sept. 13-16, 1998.

[17] B. Baidya, S.K. Gupta, T. Mukherjee, “MEMS Component Extraction,” *Proc. of MSM ‘99*, San Juan, Puerto Rico, pp. 143-6, April 19-21, 1999.

[18] C.M. Baker and C. Terman, “Tools for Verifying Integrated Circuit Designs,” *Lamda Magazine,* 4th Quarter, 1980, pp. 22-30.

[19] S.P. McCormick, “EXCL: A Circuit Extractor for Integrated Circuit Designs,” *Proceedings of the 21st DAC*, June 1984, pp. 616-23.

[20] G.M. Tarolli and W.J. Herman, “Hierarchical Circuit Extraction with detailed Parasitic Capacitance,” *ACM IEEE 20th DAC Proceedings,* 1983, pp. 734-38.

[21] T.J. Wagner, “Hierarchical Layout Verification,” *IEEE Design and Test*, February 1985, pp. 31-37.

[22] S.M. Trimberger, An Introduction to CAD for VLSI, Kluwer Academic Publisher, 1987.

[23] J.D. Ullman, Computational Aspects of VLSI, Computer Science Press, 1987.

[24] J.K. Ousterhout, “Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools,” *IEEE Transactions on Computer-Aided Design*, Vol. CAD-3, No. 1, January 1984, pp. 87-100.

[25] W.C. Tang, T.-C.H. Nguyen, M.W. Judy and R. T. Howe, “Electrostatic-comb Drive of Lateral Polysilicon Resonators,” *Transducers ‘89*, Vol. 2, pp. 328-331, June 1990.

[26] T. Hirano, T. Furuhatha, K.T. Gabriel and H. Fijita, “Design, Fabrication and Operation of Submicron Gap Comb-Drive Microactuator,” *J. of MEMS*, Vol. 1, No. 1, March 1992, pp. 52-9.

[27] W.C. Tang, M.G. Lim and R.T. Howe, “Electrostatically Balanced Comb Drive for Controlled Levitation,” *Technical Digest IEEE Solid-State Sensor and Actuator Workshop,* pp. 23-27, June 1990.